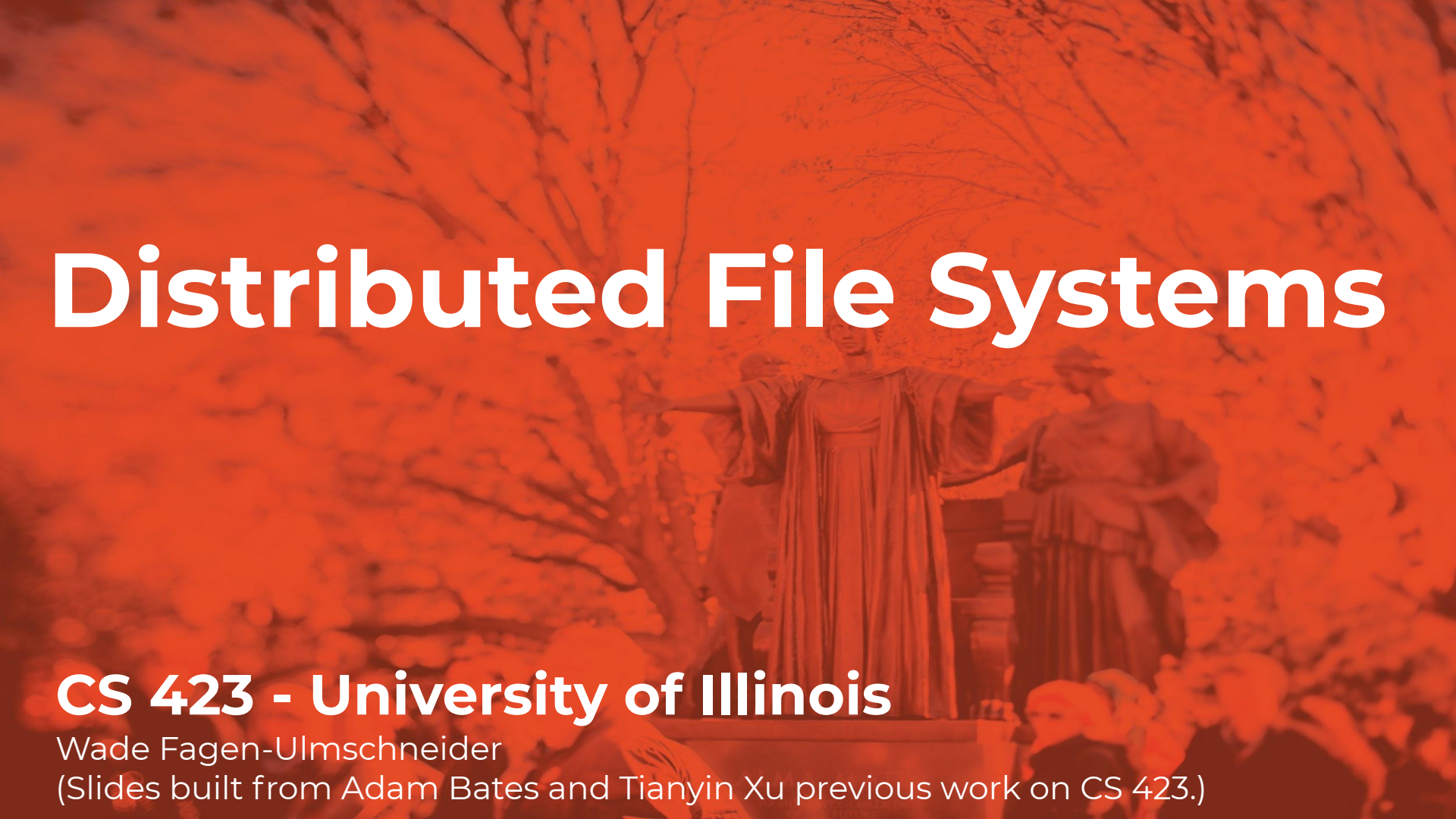


Distributed File Systems

CS 423 - University of Illinois

Wade Fagen-Ulmschneider
(Slides built from Adam Bates and Tianyin Xu previous work on CS 423.)

File Systems

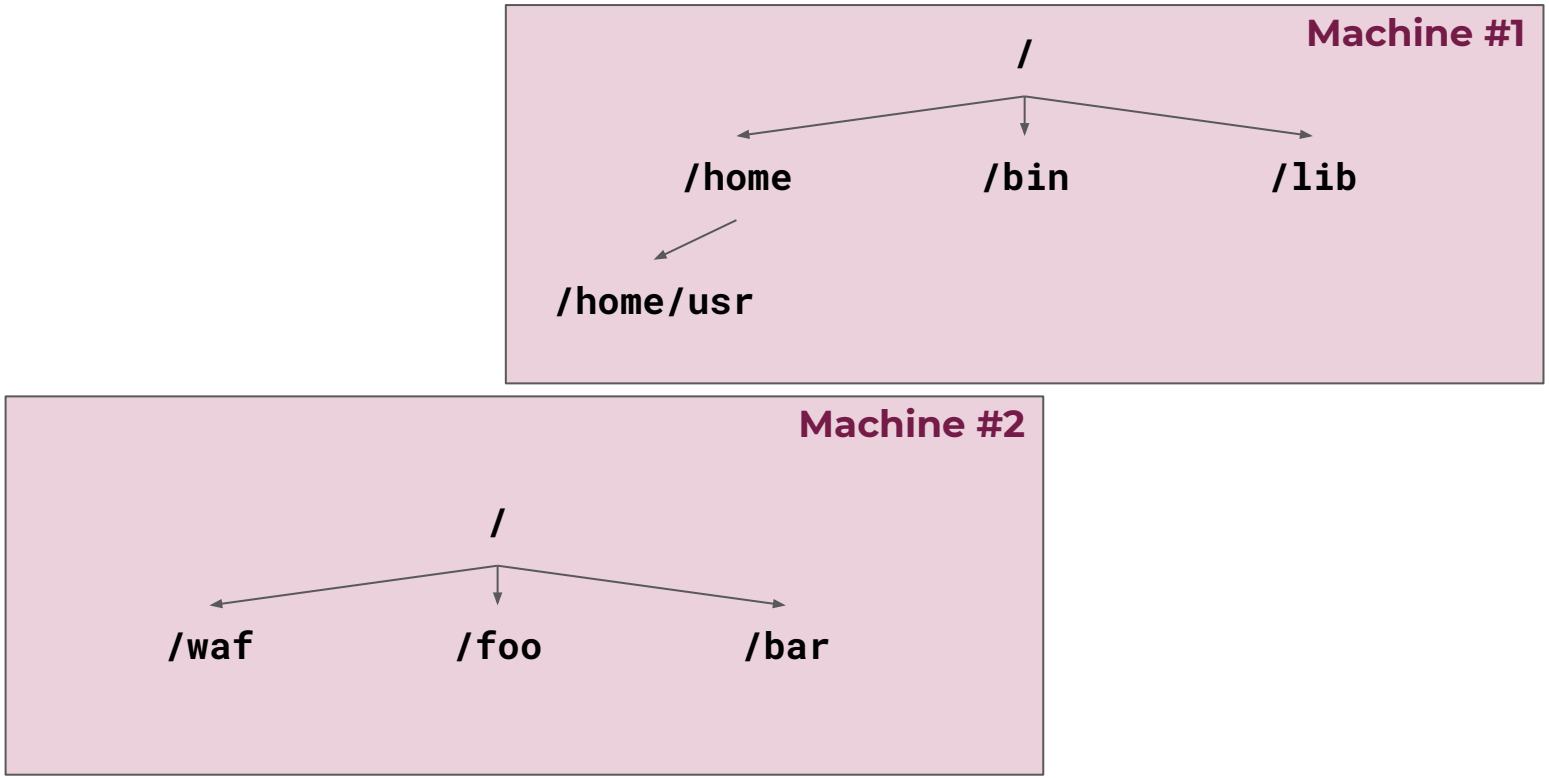
- ★ A file system provides a service for clients.
  - Provides an interface for **creating** files,
  - Provides an interface for **reading** files,
  - Provides an interface for **writing** files,
  - ...etc...

Distributed File Systems

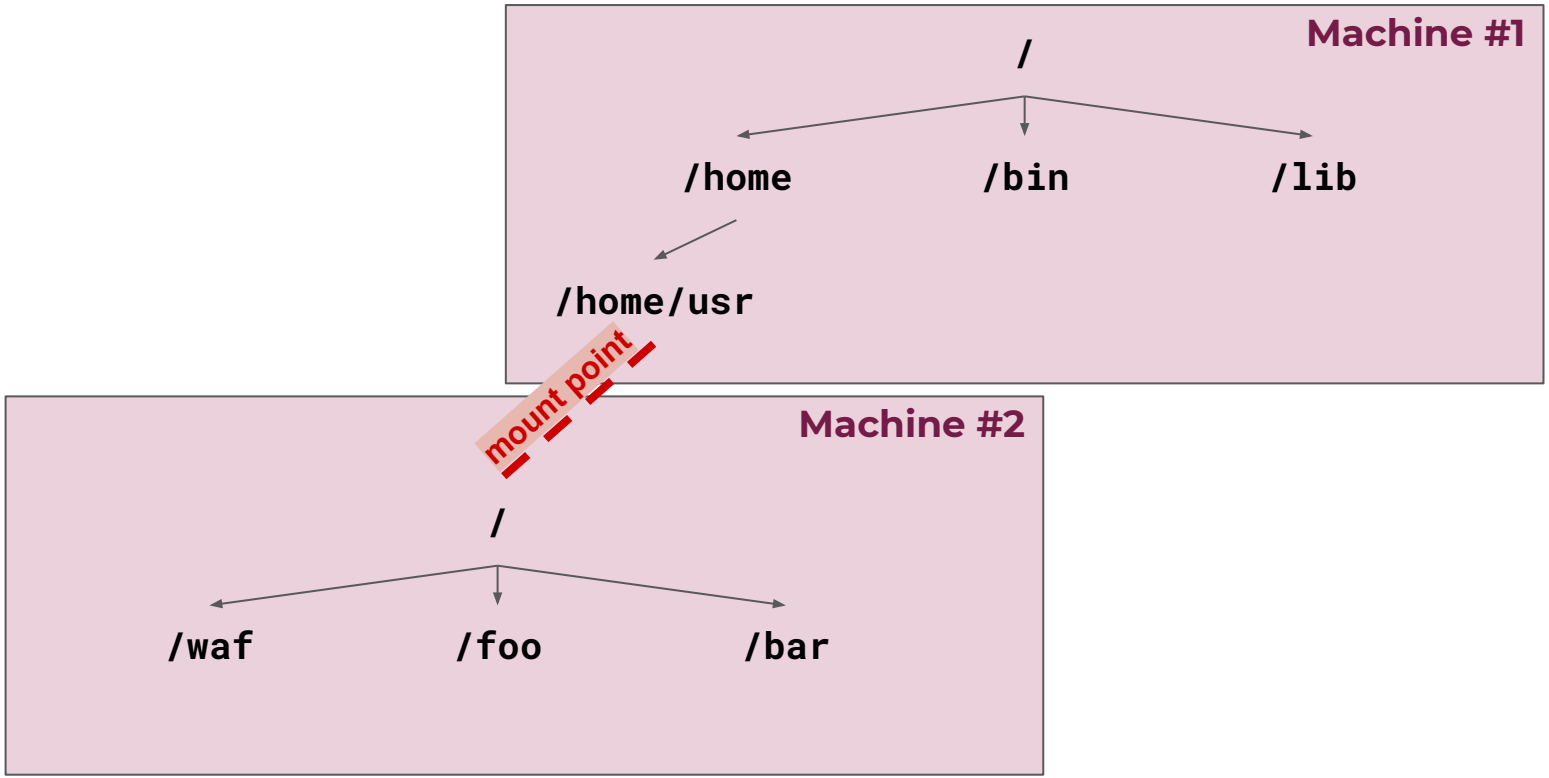
- ★ A Distributed File System (DFS) is simply a classical model of a file system distributed across multiple machines.
  - **Goal:** Share a set of dispersed files.
  - Resources on a host machine is **local**.
  - Resources on other machines is **remote**.

- ★ **NFS: Network File System** is a common DFS.

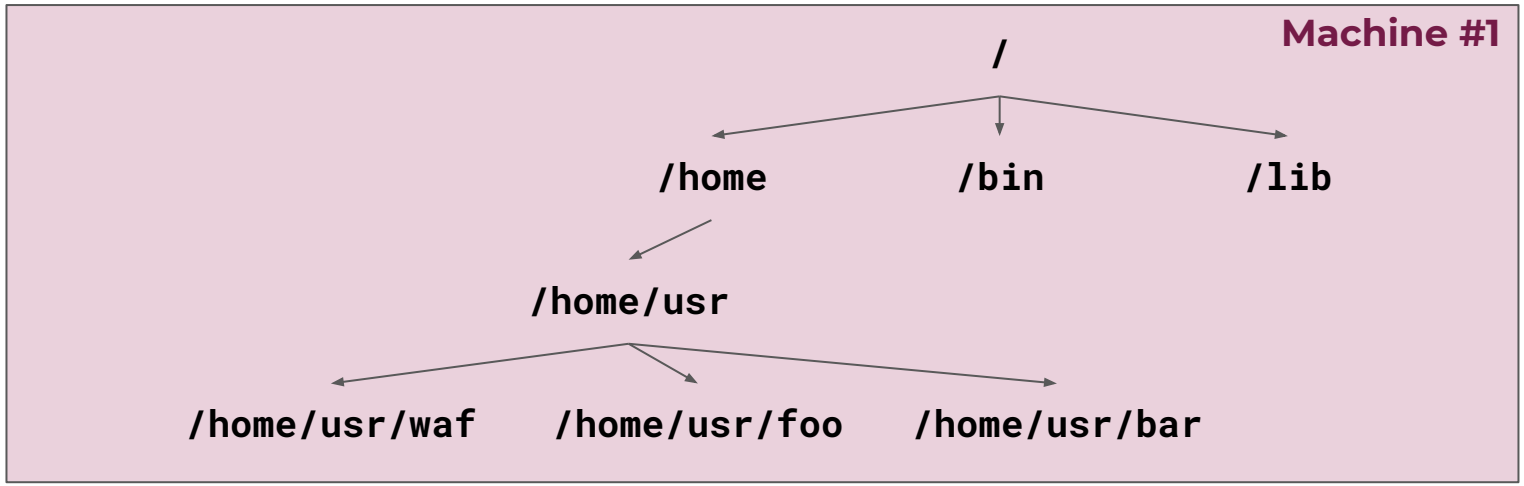
Distributed File Systems



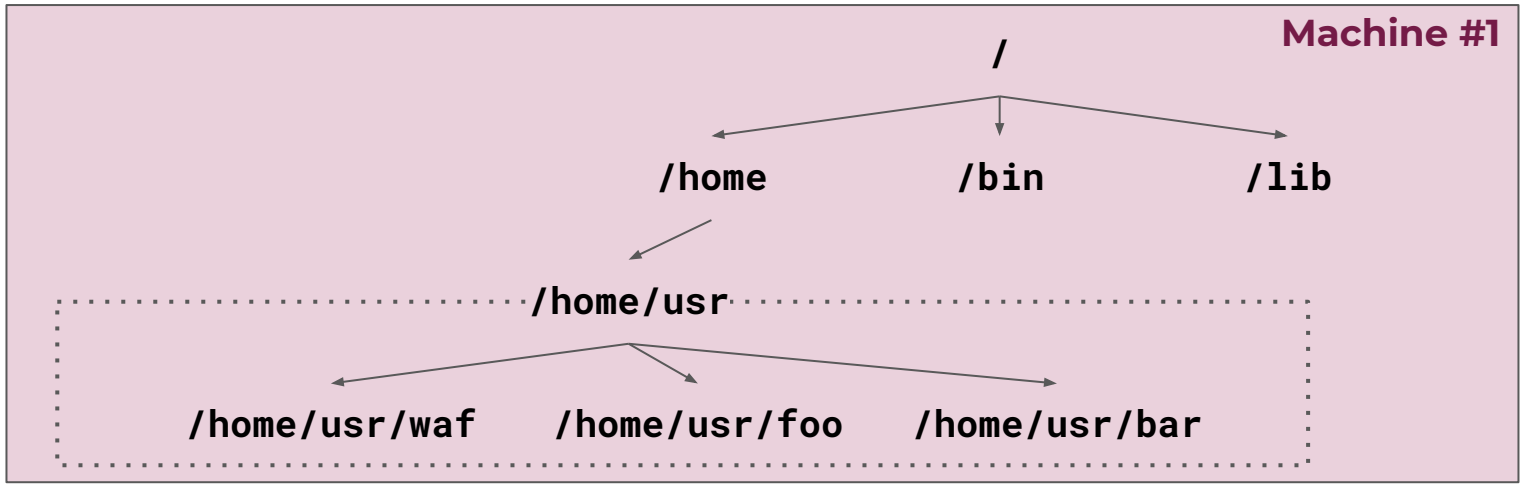
Distributed File Systems



Distributed File Systems



Distributed File Systems



★ The **logical view** of Machine #1 includes **local and remote resources**.

Caching in Distributed File Systems

- ★ The server in a DFS will nearly always be the bottleneck.
  - Idea: Increase performance using caching!

- ★ Caching Advantages:
  - Once cached, open/read/write/close can be done locally.
  - Significantly reduced network traffic.

- ★ Caching Problems:
  - Update Failures: What if the client never commits the updates to the server?
  - Consistency: Multiple clients may have different caches of a file.

NFS Overview

- ★ NFS servers are **stateless**; each request provides all arguments required for execution
  - Ex: **`ReadAt(inumber, position)`**, complete stateless NOT the standard C **read()**.
  - No need to perform network **open()** or **close()** on file.

- ★ **Idempotent**: Performing requests multiple times has same effect as performing it exactly once
  - Ex: Server crashes between disk I/O and message send, client resend read, server does operation again.
  - Ex: Read and write file blocks: just re-read or re-write file block – no side effects.
  - Ex: What about “remove”? NFS does operation twice and second time returns an advisory error.

NFS: Multiple Failure Modes

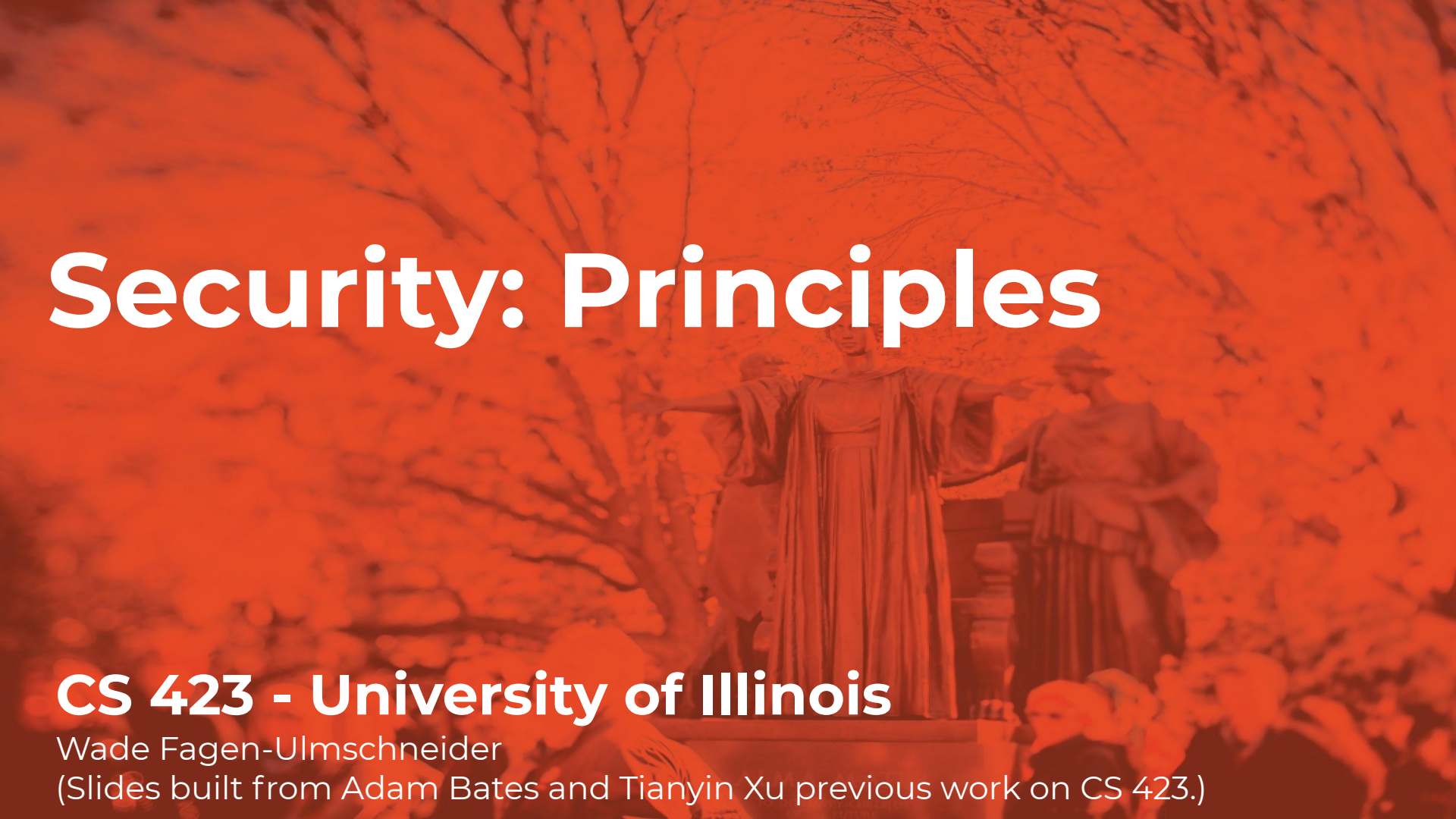
- ★ Failure Mode: Blocking
  - Block until the server comes back up.
  - *…but this may be next week?, next year?*
- ★ Failure Mode: Error
  - Return a network error to the user application.
  - *…but most applications don't even consider disk over network in their code.*

Beyond NFS

- ★ **Andrew File System (AFS), ~1980s**
  - Distributed of trusted servers as a DFS.
  - Presents a homogeneous file system across the full system of many hosts.

- ★ **Google File System (GFS), ~2010s**
  - Designed to run on cheap hardware with many failures.
  - Optimized to store large files (100s MBs+).
  - Optimized for long streaming reads (not small random reads).
  - Optimized for appended writes, not rewrites.
  - Minimizing bandwidth over minimizing latency.

CS 425: Distributed Systems



Security: Principles

CS 423 - University of Illinois

Wade Fagen-Ulmschneider
(Slides built from Adam Bates and Tianyin Xu previous work on CS 423.)

Security Properties

- ★ Confidentiality
- ★ Integrity
- ★ Authenticity
- ★ Availability

Security Properties

- **Confidentiality**
  Only trusted parties can read data.
- **Integrity**
- **Authenticity**
- **Availability**

Security Properties

★ Confidentiality

★ **Integrity**
Only trusted parties have modified data.

★ Authenticity

★ Availability

Security Properties

★ Confidentiality

★ Integrity

★ **Authenticity**
Data originates from the correct party.

★ Availability

Security Properties

★ Confidentiality

★ Integrity

★ Authenticity

★ **Availability**
Data is available to trusted parties when needed.

Security Properties

- **Confidentiality**
  Only trusted parties can read data.

- **Integrity**
  Only trusted parties have modified data.

- **Authenticity**
  Data originates from the correct party.

- **Availability**
  Data is available to trusted parties when needed.

Security Functions

- ★ **Define**
- ★ **Authentication**
- ★ **Authorization**
- ★ **Auditing**

Security Functions

- **Define** the security functions over *principals* (users, programs, admins, etc)
  - *...and also all entities (files, network sockets, IPC, etc)*
- **Authentication**
- **Authorization**
- **Auditing**

Security Functions

- ★ **Define** the security functions over *principals* (users, programs, admins, etc)

  ...and also all entities (files, network sockets, IPC, etc)

- ★ **Authentication**

  How do we determine the identity of the principal?

- ★ **Authorization**

- ★ **Auditing**

Security Functions

- **Define** the security functions over *principals* (users, programs, admins, etc)

  ...and also all entities (files, network sockets, IPC, etc)

- **Authentication**

  How do we determine the identity of the principal?

- **Authorization**

  Which principals are permitted to take what actions on which objects?

- **Auditing**

Security Functions

- **Define** the security functions over *principals* (users, programs, admins, etc)

  ...and also all entities (files, network sockets, IPC, etc)

- **Authentication**

  How do we determine the identity of the principal?

- **Authorization**

  Which principals are permitted to take what actions on which objects?

- **Auditing**

  Record of (un)authorized actions that took place on the system for post-hoc diagnostics.

Access Control Matrix

- ★ The **access control matrix** is a key feature of any authentication schema:
  - For every protected resource, list of who is permitted to do what
  - Example: for each file/directory, a list of permissions:
    - *owner, group, world*
    - *read, write, execute*
    - *setuid: program run with permission of principal who installed it*
  - Smartphone: list of permissions granted each app

Access Control Matrix

- ★ **Access control matrices** allow us to specify an arbitrary **security policy**.
  - What **properties** should our security policy provide?

Principle of Least Privilege

- ★ Grant each principal the **least permission possible for them to do their assigned work:**
  - Minimize code running inside kernel
  - Minimize code running as sysadmin

- ★ ...however, this is a hard challenge!
  - ...hard to know what permissions are needed in advance.
  - ...hard to know what permissions should be granted.
    - Ex: to smartphone apps
    - Ex: to servers

Authorization w/ Intermediaries

- ★ **Trusted Computing Base (TCB)**: set of software trusted to enforce security policy.

- ★ **Ex:** Storage Server is trusted to check user access control list
  - Why? Because server must store/retrieve data on behalf of all users.
  - Implication? security flaw in server allows attacker to take control of system

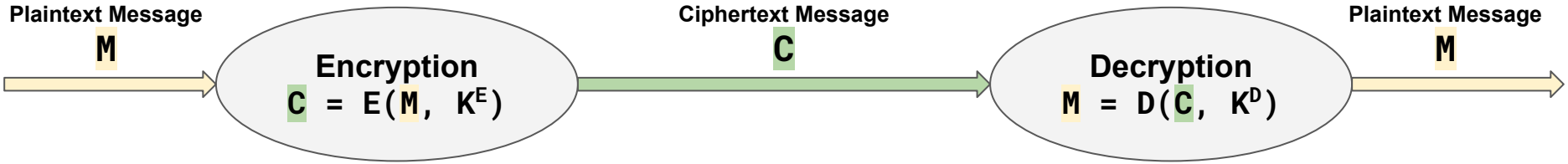
- ★ Q: Is it good or bad to have a large TCB?

Security: Encryption

CS 423 - University of Illinois

Wade Fagen-Ulmschneider
(Slides built from Adam Bates and Tianyin Xu previous work on CS 423.)

Encryption



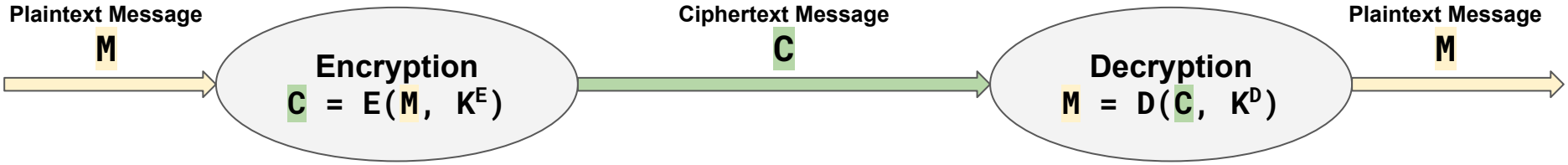
E: Encryption Function

K^E: Encryption Key (Private)

D: Decryption Function

K^D: Decryption Key (Private)

Encryption



E: Encryption Function
K^E: Encryption Key (Private)

D: Decryption Function
K^D: Decryption Key (Private)

- ★ If an attacker knows **M** (plaintext), **C** (ciphertext), **E** (encryption function), and **D** (decryption function), they should:
  - Not be able determine any private keys (K^E or K^D)
  - Not be able to modify the message
- ★ Cryptography provides basis for authentication, privacy, and integrity

Authentication: Password

- ★ **Q:** How do we know user is who they say they are?

- ★ With password-based authentication, user shares a “private” secret (their password). However:
  - User must remember their password
  - Short passwords ⇒ easy to remember, easy to guess!
  - Long passwords ⇒ hard to remember

Authentication: Password

- **Q:** How do we know user is who they say they are?

- With password-based authentication, user shares a “private” secret (their password). However:
  - User must remember their password
  - Short passwords ⇒ easy to remember, easy to guess!
  - Long passwords ⇒ hard to remember

  - ***Q: How do we store passwords anyhow?***

Storing Passwords

- ★ Store passwords in a file/database?
  - Anyone with sysadmin rights can read the passwords!

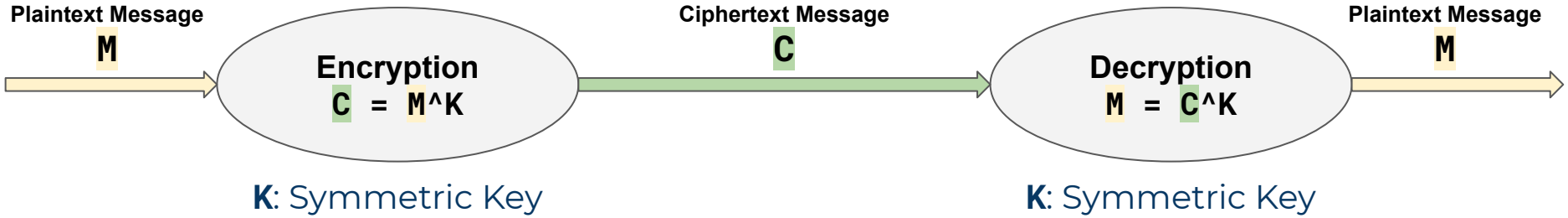
- ★ Encrypt passwords in a file/database?
  - If gain access to file/database, can check passwords offline.
  - If user reuses password, easy to check against other systems.

- ★ Encrypted in a file/database with a random salt?
  - Storage := HASH(Password ^ Salt)
  - Protects against a precomputed password table lookup

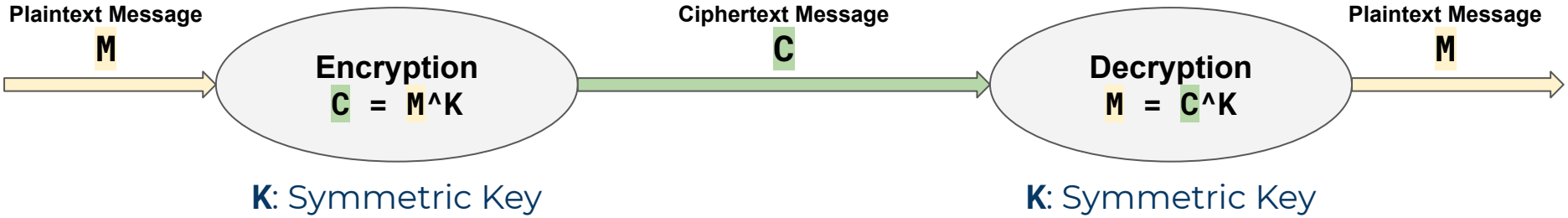
Authentication: Password

★ Passwords can be thought of as a primitive form of **symmetric key encryption**:

- K^E (encryption key) and K^D (decryption key) are identical, **K**.



Authentication: Password



- ★ If K is secure, both parties know M is authentic and secret.
- ★ Symmetric Key Examples: DES, AES

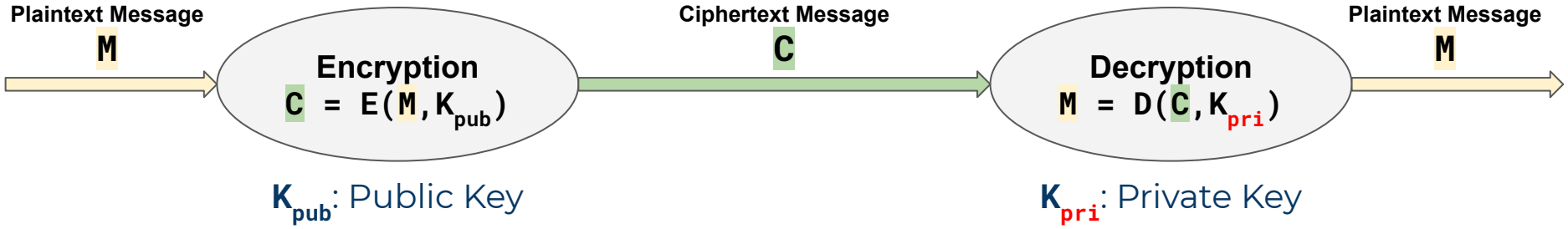
Authentication: Private Key

- ★ **Q:** How do we know user is who they say they are?

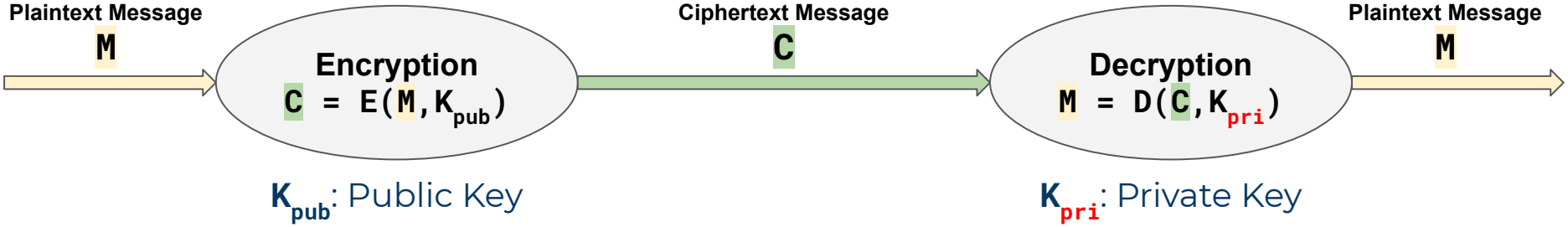
- ★ With private-key authentication, user has a file that stores a long, cryptographic key (ex: 2048 bits).
  - User needs to safely store this secret!
  - Is the system storing the key secure?
  - How do we prove the secret without revealing details of the secret?

Authentication: Private Key

- ★ Private Key Encryption provides asymmetric encryption:
  - K_{pub} (public key), available and widely accessible to everyone
  - K_{pri} (private key), private to the user



Authentication: Private Key



★ Keys are generated in pairs (K_{pub}, K_{pri}) and K_{pri} is kept private.

★ Only a private key holder (K_{pri}) can read the ciphertext message **C**.
- *Ensures secrecy of the message.*

Two-Factor Authentication

- ★ **Fact:** Long cryptographic keys are hard to manage, can we get the best of both worlds?

- ★ Store the private key ($\mathbf{K_{pri}}$) inside of a chip.
  - Use a password/PIN to authorize access to the cryptographic key.
  - Use challenge/response to authenticate smartcard.
  - *...or other methods...*

Public Key to Single Use Session Key

- ★ **Fact:** Public key encryption/decryption is slow; so can use public key to establish (shared) session key.

- ★ Use public/private key to share a single use **session key**:
  - Unique session key is generated for a single session.
  - Provides the security advantages of public/private key while the simplicity and speed of symmetric encryption.

Federated Authentication

- ★ In large networks, infeasible for everyone to share a secret with everyone else.
  - **Solution:** “Authentication Server” (Kerberos)

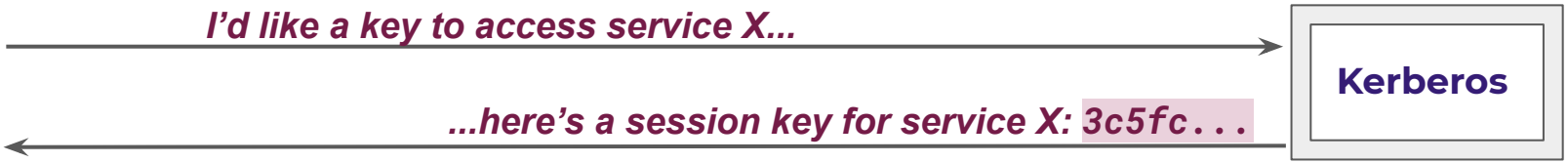
  - Everyone shares (a separate) secret with a Kerberos server.
  - Server provides shared session key for the service requested.
  - Everyone trusts authentication server.
    - *However, if compromise server, can do anything!*

Federated Authentication

★ In large networks, infeasible for everyone to share a secret with everyone else.

- **Solution:** “Authentication Server” (Kerberos)

- Everyone shares (a separate) secret with a Kerberos server.
- Server provides shared session key for the service requested.
- Everyone trusts authentication server.
  - *However, if compromise server, can do anything!*



CS 461: Computer Security